Journal of Technology Informatics and Engineering (JTIE)
Vol. 3 No. 2 August 2024
E-ISSN: 2987-9434; P-ISSN : 2988-0343, Pages 188-202

Error-Free Arduino Communication: Integrating Hamming Code for UART Serial Transmission

Budi Raharjo[1], Fujiama Diapoldo Silalahi[1]
Email: budiraharjo@stekom.ac.id, fujiama@stekom.ac.id,
Orcid: 0000-0001-6192-0888, 0009-0007-2162-2051,
[1]*University of Science and Computer Technology, Semarang, Indonesia, 50192*
*Corresponding Author

Abstract
Serial communication is a fundamental method for data transfer in electronic devices, particularly in Arduino-based systems. However, existing protocols, such as Universal Asynchronous Receiver/Transmitter (UART), often lack robust error detection mechanisms, leading to potential data integrity issues. This study aims to address the knowledge gap regarding error detection in UART communication by implementing Hamming Code, a well-established method for detecting and correcting single-bit errors. The research employs a systematic approach, including data encoding before transmission and decoding with error correction at the receiver end. The results demonstrate that the integration of the Hamming Code significantly enhances the reliability of data transmission, reducing error rates and improving overall system performance. The implications of this research extend to various applications requiring high data integrity, such as industrial control systems and Internet of Things (IoT) devices. By providing a practical solution to the challenges of error detection in serial communication, this study contributes to the advancement of reliable communication systems in modern technology.

Keywords: *Serial Communication, Hamming Code, Error Detection, Arduino, Data Integrity.*

I. INTRODUCTION

Serial communication is an essential method for data transfer between electronic devices, including Arduino-based systems (Chioran & Valean, 2020; Jamdar et al., 2018). This method is widely utilized due to its efficiency in transmitting data through a simple communication channel. One popular serial communication protocol is the Universal Asynchronous Receiver/Transmitter (UART), which facilitates data transmission and reception without the need for external synchronization (Gupta & Charan, 2024; Jamdar et al., 2018; Lin, 2020). However, despite its convenience, UART transmission processes are often vulnerable to interference or noise, which can result in errors in the received data.

Errors in data transmission can lead to significant damage, especially in applications that require high accuracy and data integrity, such as industrial control systems, medical devices, and Internet of Things (IoT) applications (Misra et al., 2022; Serror et al., 2021). Many current serial communication systems lack built-in mechanisms to detect or correct bit errors in transmitted data. This gap in reliability can lead to serious functional failures due to undetected errors.

Received on June 27, 2024; Revised July 02, 2024; Accepted July 29, 2024; Published on August 21, 2024
Doi: 10.51903/jtie.v3i2.187

To address this issue, this study aims to C (Mythry et al., 2024; Salamea et al., 2020) in UART-based serial communication on the Arduino platform. The Hamming Code was selected due to its proven ability to efficiently detect and correct single-bit errors (Baviera et al., 2020; Potestad-Ordóñez et al., 2020; Xie et al., 2023). The implementation of this method includes data encoding before transmission, as well as decoding and error correction at the receiver. Consequently, it is anticipated that the application of the Hamming Code will enhance the reliability of serial communication and reduce error rates in data transmission. The contribution of this research lies in the development of a more robust and reliable communication system, providing a practical solution to improve the quality and reliability of transmitted data. Additionally, the results of this study are expected to offer valuable insights for further advancements in information and communication technology, particularly in applications requiring high data accuracy.

II. LITERATURE REVIEW

A. Error Detection

Error detection is a critical process aimed at identifying and addressing errors that arise within a system. These errors can stem from various sources, such as human mistakes, hardware failures, or software bugs. Effective error detection is essential to ensure that the system operates correctly and delivers the expected outcomes (Avižienis et al., 2004). There are several types of errors (Jeffries et al., 2022), including syntax errors, which occur due to mistakes in the grammar or structure of the code, such as unmatched parentheses or misspelled keywords. Logical errors arise when the algorithm fails to produce the correct output, as seen in faulty if-else conditions. Runtime errors manifest during program execution, like division by zero or accessing an invalid array index.

Error detection methods encompass several approaches. Testing involves techniques such as unit testing, integration testing, and system testing to identify code errors. Debugging focuses on locating and fixing errors by enabling developers to step through code and inspect variable values at each step. Logging involves adding log statements to the code to track program execution and help identify the source of errors. Validation ensures that the product meets user requirements, while verification ensures that the product is built correctly according to specifications. Tools like lint, static code analysis tools, and dynamic analysis tools assist in identifying errors before and after compilation.

The benefits of error detection are manifold. It enhances system quality by identifying and correcting errors, leading to an improved user experience. Early error detection reduces long-term maintenance costs. Additionally, error detection improves security by identifying vulnerabilities that malicious actors could exploit. A system free from errors is generally more reliable and trustworthy over time.

However, error detection is not without its challenges. Complex systems with numerous interacting components make error detection more difficult. Some errors may be hidden and only surface under specific conditions, making them hard to reproduce. Furthermore, certain error detection techniques may impact system performance, necessitating a balance between error detection and system efficiency. By employing appropriate techniques and tools, error detection can be conducted effectively, ensuring that the system functions as intended and reducing the likelihood of future issues.

B. Error Correction

Error detection involves the identification and resolution of issues that arise within a system. These errors can originate from various sources, including human mistakes, hardware failures, or software bugs. The process is critical in ensuring that the system operates properly and produces the desired output (Banerjee et al., 2021; Chandra & Toueg, 1996). Errors can be categorized into several types, as syntax errors occur when there is a mistake in the programming language rules or code structure, such as unmatched parentheses or incorrect keyword spelling. Logical errors arise when the algorithm fails to produce the correct outcome, as seen in incorrect if-else conditions. Runtime errors happen during the program's execution, such as dividing by zero or accessing an invalid array index.

Several techniques are utilized for error detection. Testing involves various methods, such as unit testing, integration testing, and system testing, to detect errors in the code. Debugging is the process of finding and correcting errors within the code. Debugging tools allow developers to step through the code and inspect variable values at each stage. Logging involves including log statements in the code to track program execution and help identify where errors occur. Validation ensures that the product meets user requirements, while verification ensures that the product is built according to specified standards. Tools such as lint, static code analysis tools, and dynamic analysis tools assist in detecting errors before and after compilation.

The benefits of error detection are significant. By identifying and correcting errors, the overall quality of the system improves, resulting in a better user experience. Detecting and fixing errors early in the development cycle reduces long-term maintenance costs. Error detection also enhances security by identifying vulnerabilities that malicious actors could exploit. A system free from errors is generally more dependable in the long term. However, error detection presents several challenges (Nsaif et al., 2021; Yu & Zhang, 2022). Complex systems with many interacting components make error detection more difficult. Some errors may not be immediately visible and only emerge under specific conditions that are hard to reproduce. Additionally, some error detection techniques can affect system performance, necessitating a balance between error detection and performance. By leveraging appropriate techniques and tools, error detection can be carried out effectively, ensuring the system functions as expected and minimizing the likelihood of future issues.

C. Arduino Serial

Arduino Serial refers to the communication method between an Arduino board and a computer or other devices through a serial interface (Asadi, 2023; Maemunah & Riasetiawan, 2018). This technique is crucial and frequently used in data communication, programming, and debugging for Arduino projects (Qurrata, 2018). The basics of Arduino Serial communication include several key components. The serial interface allows data to be sent one bit at a time over a communication line. On Arduino, this communication is typically conducted through a USB port connected to a computer. Arduino is equipped with a serial port that can be utilized for interacting with other devices or programming the board itself. The baud rate measures the data transmission speed in bits per second (bps), and the baud rate needs to be consistent between devices to ensure proper data transfer. Common baud rates include 9600, 115200, and 19200.

Arduino Serial serves several primary functions. First, it is used for programming, where the serial interface uploads programs from the computer to the Arduino board. During programming through the Arduino IDE, data is transmitted to the board via serial communication. Second, it facilitates data communication between Arduino and the computer, which is particularly useful for applications such as sending sensor data to a computer or remotely controlling Arduino. Third, serial communication is a valuable tool for debugging, allowing messages to be sent via the serial port. The code output can be viewed in the Arduino IDE's Serial Monitor, aiding in identifying and resolving errors.

Key methods used in Arduino Serial include the following. The `Serial.begin()` function initiates serial communication at a specified baud rate. For instance, `Serial.begin(9600);` starts communication at 9600 bps. The `Serial.print()` and `Serial.println()` functions send data from Arduino to the computer, with the latter adding a new line at the end of the data. The `Serial.read()` function reads data received through the serial port, often used to read input from the computer or other devices. The `Serial.available()` function provides the number of bytes of data ready to be read from the serial port. Lastly, the `Serial.flush()` function clears any data remaining in the serial buffer. The advantages of using Arduino Serial include ease of debugging, as it allows developers to check variable values and code execution flow during development (Kondaveeti et al., 2021; Martínez-Santos et al., 2017). It also provides efficient and simple data communication between

Arduino and other devices. Additionally, Arduino Serial offers flexibility, supporting a wide range of applications from sensor monitoring to device control through serial input.

However, using Arduino Serial presents certain challenges (Rafi, 2019). Ensuring the correct baud rate is crucial, as mismatched baud rates can result in unreadable data. The limited size of the serial buffer may lead to data loss if the buffer overflows. Moreover, signal interference and noise in complex serial communication can affect communication quality. Arduino Serial is a highly valuable and flexible tool for Arduino-based project development. With a solid understanding of its fundamental concepts and techniques, you can leverage serial communication to enhance the performance and effectiveness of your projects (Muhajir, F., Efendi, S. & Sutarman, 2016).

D. Universal Asynchronous Receiver-Transmitter (UART)

UART (Universal Asynchronous Receiver-Transmitter) is a serial communication method used to transfer data between devices. UART operates by transmitting data sequentially, bit by bit, through a single communication channel, making it one of the most widely implemented serial communication methods in computer systems and electronic devices (Agus, 2019). The operation of UART involves asynchronous transmission, where data is sent without an external clock signal. The transmission speed, or baud rate, must be agreed upon between the transmitting and receiving devices to ensure proper synchronization of communication. Data is transmitted in a frame format consisting of several components: the start bit, which indicates the beginning of data transmission; data bits, which form the core of the frame and typically consist of 7 or 8 bits; an optional parity bit, used for error detection and can be set to check for even or odd parity; and one or more stop bits, which signify the end of the data frame (Mukti, R., & Fadilah, 2018).

In UART communication, the transmitter sends data by converting parallel data from the CPU or system into a serial format for transmission. The receiver then converts the serial data back into parallel format for processing by the CPU or system. Key UART settings include baud rate, which measures data transmission speed in bits per second (bps) and must match between devices; data bits, usually 7 or 8 bits per frame; parity, an optional error-checking method that can be set to none, even, or odd; and stop bits, typically one or two, which allow extra time for synchronization between data frames. UART offers several advantages, including simplicity, as it is easy to implement, making it ideal for applications that do not require high data transmission speeds. Its cost-effectiveness also makes it a cheaper option compared to more complex serial communication protocols. Additionally, UART is flexible and can be used in a variety of applications, from microprocessor communication to peripheral connections.

However, UART has some limitations. It is less suitable for applications requiring very high data transmission speeds, as the speed is constrained by the baud rate that devices can handle. The absence of a clock signal in this asynchronous protocol can lead to data transmission errors if synchronization is not accurate. Furthermore, UART is more effective for short-distance communication, as signal issues and interference can pose challenges over longer distances (Kusnadi, 2021). UART is frequently used for communication between microprocessors or microcontrollers in embedded systems, connecting peripherals such as sensors or GPS modules to computers or microcontrollers, and enabling wireless communication through Bluetooth and Wi-Fi modules. Despite its limitations in speed and range, UART remains a popular choice for communication systems due to its ease of implementation and low cost (Novianti, 2019).

III. RESEARCH METHOD(S)

A. *System Design*

The methodology begins with system design, which involves selecting the primary components: the Arduino Uno and the DHT11 sensor. This design encompasses a schematic representation illustrating how the sensor is connected to the Arduino and how temperature and humidity data will be collected and processed.

B. *Implementation of Hamming Code*

The implementation of the Hamming Code is divided into two main parts: the transmitter and the receiver. For the transmitter, the encoding process involves adding parity bits at specific positions within the data array. These parity bits are computed using XOR logic operations on the relevant data bits. An encoding flowchart is constructed to depict the steps in this process, including the determination of parity bit positions and the calculation of their values. For the receiver, after the data is transmitted, the received codeword is stored in an array variable. The decoding process is then carried out to verify data integrity and correct any detected errors. The steps involved in decoding are also illustrated in a flowchart to provide a clear overview of the workflow.

C. *Testing and Analysis*

Testing is conducted to evaluate the system’s performance in both functional and non-functional aspects. Functional testing involves measuring the accuracy of temperature readings and the conversion of data into a 12-bit format. Results are displayed on the Arduino IDE serial monitor for further analysis. Non-functional testing covers aspects such as processing speed, system reliability, and the effectiveness of the encoding process. This testing also includes modifying encoded data to assess the system's capability to detect and correct errors.

D. Evaluation of Results

The results from functional and non-functional testing are analyzed to ensure that the system meets the established requirements. Data obtained from these tests are compared against expected standards to evaluate overall system performance.

IV. RESULT/FINDINGS AND DISCUSSION

A. System Design

System design involves creating an effective and efficient information system to support organizational goals. This process encompasses several critical stages, including needs analysis, implementation, and maintenance. During this phase, information is gathered from users and stakeholders to understand business needs and identify the necessary features for the system. The system architecture is then designed, incorporating technical specifications and user interface designs. This design phase also includes the development of flowcharts, data models, and other detailed specifications. By following these steps, system design helps organizations achieve operational efficiency and support the attainment of strategic objectives.

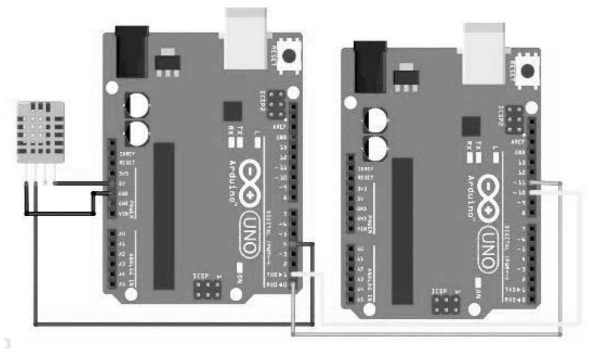


Figure1. Schematic of the DHT11 sensor and Arduino Uno design

B. Hamming Code Design for the Transmitter

The design of the Hamming Code for the transmitter ensures that the system can meet its non-functional requirements, particularly its capability for data encoding. This encoding process generates codewords that include the necessary parity bits to detect and correct errors. In the Hamming Code, parity bits are added at specific positions in the array, such as indices 0, 1, 3, 7, and x. These positions are selected based on powers of two, allowing the method to detect errors in more than one bit, including two-bit errors, through XOR operations. By calculating each parity bit from specific data bits and utilizing XOR, the system ensures that errors during data transmission can be detected and corrected.

The encoding process on the transmitter side involves determining the positions and calculations of parity bits and placing them in the designated positions. These parity bits are used to verify the integrity of the transmitted data, enabling the receiver to detect and correct errors, thereby ensuring that the received data remains accurate and intact.

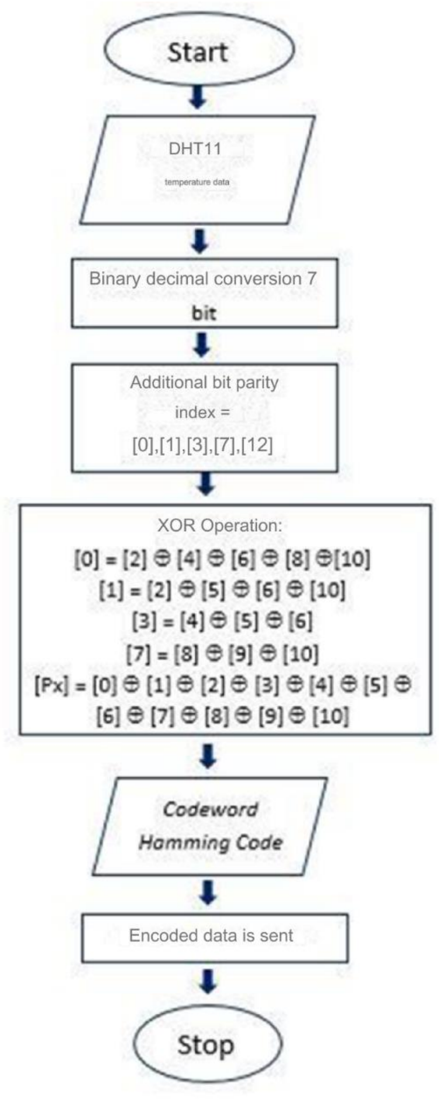


Figure 2. Hamming Code Encoding Flowchart for the Transmitter

C. *Hamming Code Design for the Data Receiver*

The received codeword is stored in an array variable, with the array length adjusted to match the number of data bits received from the encoding process. The decoding process using the Hamming Code involves storing the data, error checking, error correction, and separating the codeword from the original data. Initially, the received codeword is stored in an array variable according to the number of bits encoded during the encoding process. Next, error checking is performed using the Hamming Code method, which involves recalculating the parity bits and comparing them with the received parity bits to detect errors.

If errors are detected, the position of the erroneous bit is identified based on the parity bit calculations, and the error is corrected by flipping the value of the erroneous bit. After error correction, the parity bits are removed, leaving only the original data. The corrected codeword and original data are then separated for the next steps. The corrected original data is converted from binary to decimal format, and the required temperature or other data is displayed based on the conversion results. This decoding process ensures that the received data is error-free and ready for use.

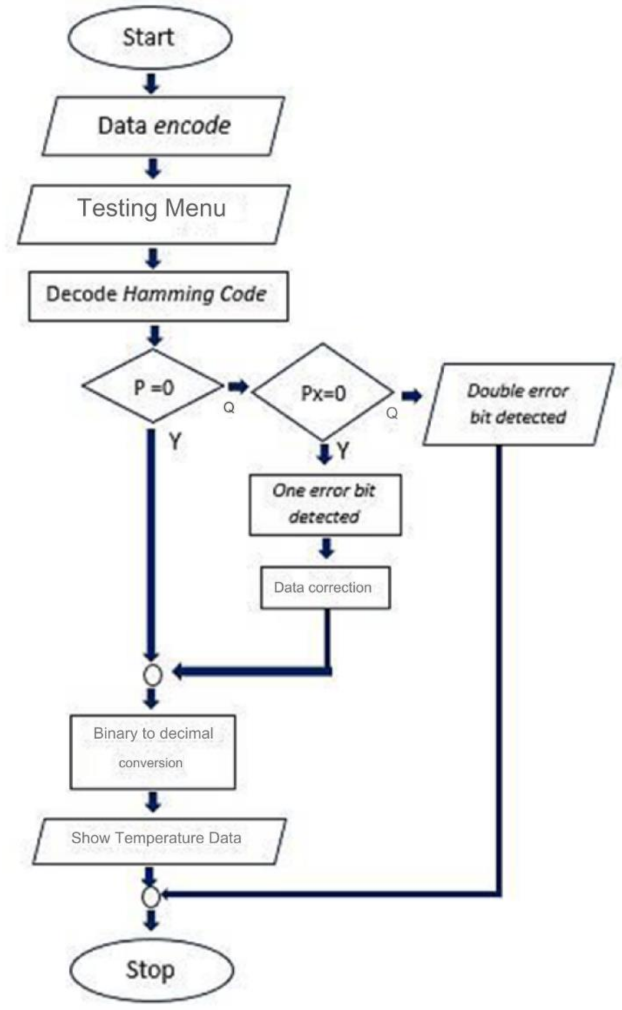


Figure 3. Decoding Flowchart for the Receiver

D. *Implementation*

Implementing the Hamming Code on the Arduino Uno provides an efficient method for detecting and correcting errors in digital data transmission. This system utilizes two Arduino Uno microcontrollers connected as a transmitter and receiver. On the transmitter side, the Arduino Uno is equipped with a DHT11 sensor that measures temperature and humidity. The data collected from this sensor is encoded using the Hamming Code, which adds parity bits to detect and correct errors. The encoded data is then transmitted via serial communication to the receiving Arduino.

On the receiver side, the Arduino Uno receives the transmitted data and decodes it using the Hamming Code to identify and correct any errors that may have occurred during transmission. The corrected data is then displayed or utilized according to application needs. This implementation enhances the reliability of the communication system, ensuring that transmitted data is more accurate, particularly in applications requiring environmental monitoring such as temperature and humidity. The Hamming Code method can be applied to various applications requiring reliable and error-free data transmission.

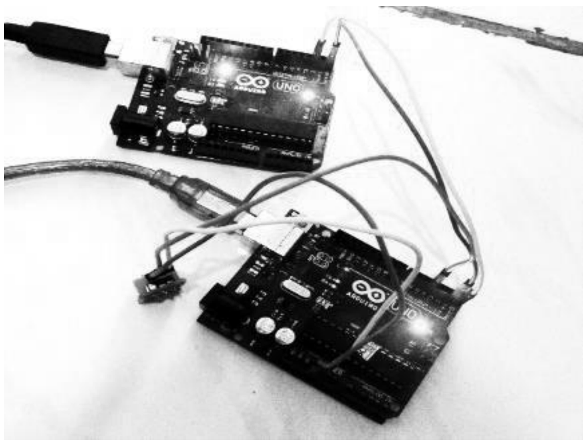
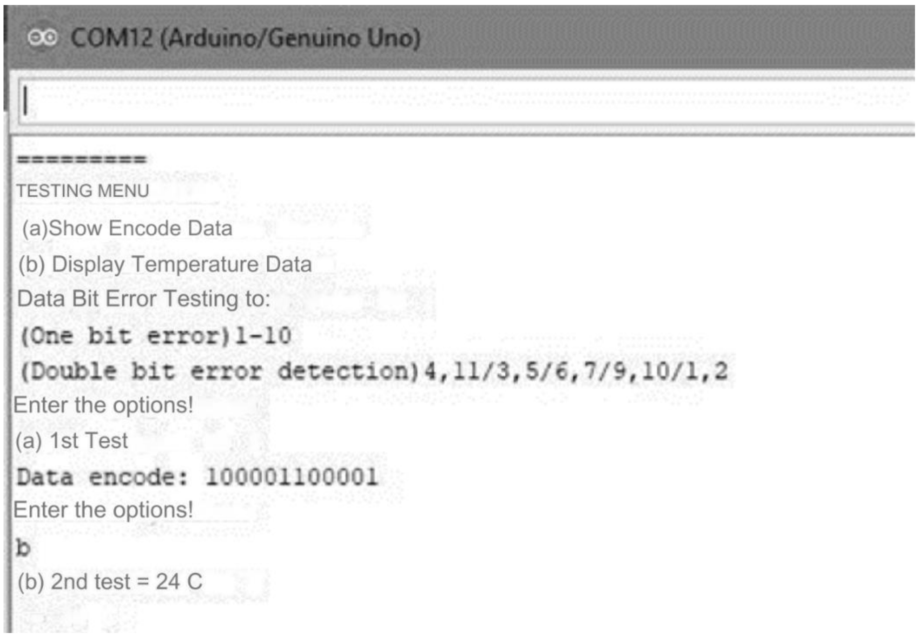


Figure 4. Implementation of the Arduino Uno and DHT11

E. Testing and Analysis

1. Functional and Non-Functional Testing

Testing evaluates both functional and non-functional requirements based on the design and implementation results. Functional requirements encompass the expected features and functions operating according to the initial design, while non-functional requirements cover aspects such as performance, reliability, and system efficiency. The goal of this testing is to analyze whether the system operates as intended. Results will be analyzed to ensure that functional requirements are met and that system performance aligns with expected standards. The testing results displayed on the Arduino IDE serial monitor, as shown in the figure below, illustrate how data and testing information are presented, providing insights into system performance and implementation effectiveness.



```
COM12 (Arduino/Genuino Uno)

=========
TESTING MENU
 (a)Show Encode Data
 (b) Display Temperature Data
 Data Bit Error Testing to:
 (One bit error)1-10
 (Double bit error detection)4,11/3,5/6,7/9,10/1,2
Enter the options!
(a) 1st Test
Data encode: 100001100001
Enter the options!
b
(b) 2nd test = 24 C
```

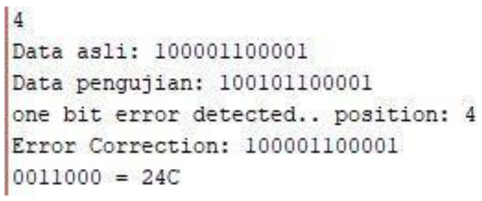
Figure 5. Testing Menu

Based on the design and implementation, functional and non-functional testing was conducted with the temperature set at 24°C, and the temperature data encoded in a 12-bit format. The results of this process are illustrated in Figure 7, which shows how the encoded temperature data is displayed and processed during testing. Functional testing focuses on verifying whether the system accurately measures temperature and converts the data into the 12-bit format as planned. Non-functional testing includes evaluating other aspects such as processing speed, system reliability under these conditions, and the effectiveness of the encoding process. Figure 7 provides a visualization of the testing results, demonstrating how the encoded temperature data is displayed and processed, allowing for an assessment of system performance and accuracy.

```
(b) Testing-5

=24 C

(c) Testing-6

Data encode: 100001100001

Insert choice!

...
```

Figure 6. Testing Data

The figure below displays the system's capability to meet functional and non-functional requirements. Functional requirements involve the system's ability to accurately measure temperature data, while non-functional requirements pertain to encoding temperature data using the Hamming Code, aimed at enhancing data reliability by detecting and correcting errors that may arise during transmission. Subsequent testing involves modifying the encoded data by selecting random positions. The objective of this process is to test the system's ability to detect and correct errors intentionally introduced into the encoded data. By altering data at random positions, this testing evaluates the effectiveness of the Hamming Code in maintaining data integrity and the system's capability to handle and correct errors. The figure below provides an overview of how the system manages encoded data and tests its robustness against introduced errors.

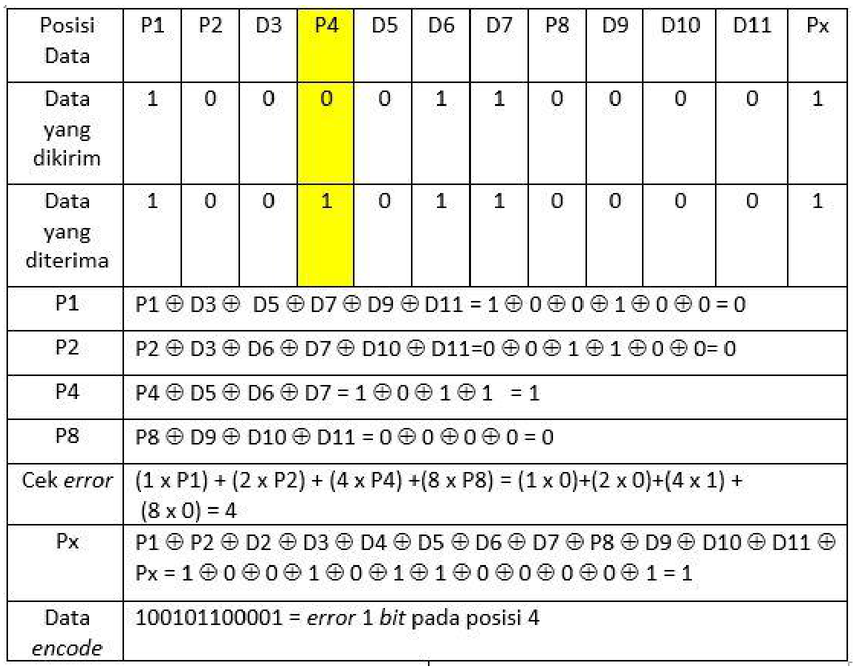


```
4
Data asli: 100001100001
Data pengujian: 100101100001
one bit error detected.. position: 4
Error Correction: 100001100001
0011000 = 24C
```

Figure: Testing 1-bit Error at Position 4

In the figure, a single bit of data is altered at position 4, causing an error at that bit. The system is capable of detecting the type of error and correcting the affected bit, thereby maintaining data integrity despite transmission errors. The table below will present a manual calculation using the Hamming Code method to detect and correct errors in temperature data set at 24°C. The table will outline the steps taken in the error detection and correction process, demonstrating how the Hamming Code method can be used to identify and correct errors in encoded data.

Table: Manual Calculation Using the Hamming Code Method for 24°C Data.



Posisi Data	P1	P2	D3	P4	D5	D6	D7	P8	D9	D10	D11	Px
Data yang dikirim	1	0	0	0	0	1	1	0	0	0	0	1
Data yang diterima	1	0	0	1	0	1	1	0	0	0	0	1
P1	P1 ⊕ D3 ⊕ D5 ⊕ D7 ⊕ D9 ⊕ D11 = 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 ⊕ 0 = 0											
P2	P2 ⊕ D3 ⊕ D6 ⊕ D7 ⊕ D10 ⊕ D11=0 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 ⊕ 0= 0											
P4	P4 ⊕ D5 ⊕ D6 ⊕ D7 = 1 ⊕ 0 ⊕ 1 ⊕ 1 = 1											
P8	P8 ⊕ D9 ⊕ D10 ⊕ D11 = 0 ⊕ 0 ⊕ 0 ⊕ 0 = 0											
Cek *error*	(1 x P1) + (2 x P2) + (4 x P4) +(8 x P8) = (1 x 0)+(2 x 0)+(4 x 1) + (8 x 0) = 4											
Px	P1 ⊕ P2 ⊕ D2 ⊕ D3 ⊕ D4 ⊕ D5 ⊕ D6 ⊕ D7 ⊕ P8 ⊕ D9 ⊕ D10 ⊕ D11 ⊕ Px = 1 ⊕ 0 ⊕ 0 ⊕ 1 ⊕ 0 ⊕ 1 ⊕ 1 ⊕ 0 ⊕ 0 ⊕ 0 ⊕ 0 ⊕ 1 = 1											
Data *encode*	100101100001 = *error* 1 *bit* pada posisi 4											

V. CONCLUSION AND RECOMMENDATION

This research successfully demonstrates the integration of Hamming Code into UART serial communication for Arduino systems, addressing the critical issue of data integrity in electronic communications. The implementation of the Hamming Code has been shown to significantly reduce error rates during data transmission, thereby enhancing the reliability of the system. The findings indicate that by incorporating error detection and correction mechanisms, the performance of Arduino-based applications can be improved, making them more suitable for environments where data accuracy is paramount, such as in industrial automation and IoT applications.

Recommendations for Future Research

Future research should explore the scalability of the Hamming Code implementation in more complex communication systems, including multi-device networks and higher data rates. Additionally, investigating alternative error correction codes, such as Reed-Solomon or Turbo Codes, could provide insights into further enhancing data integrity. It would also be beneficial to conduct real-world testing in various environmental conditions to assess the robustness of the proposed solution. Finally, integrating machine learning techniques for adaptive error correction could be a promising avenue for future studies, potentially leading to smarter and more resilient communication systems.

REFERENCES

Agus, F. (2019). Sistem Komunikasi dan Pengkodean Data pada Arduino . *Jurnal Ilmu Komputer, 15(4), 25-33*.

Asadi, F. (2023). *Serial Communication*. 179–199. https://doi.org/10.1007/978-1-4842-9600-4_5

Avižienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, *1*(1), 11–33. https://doi.org/10.1109/TDSC.2004.2

Banerjee, S., Samynathan, B., Abraham, J. A., & Chatterjee, A. (2021). Real-Time Error Detection in Nonlinear Control Systems Using Machine Learning Assisted State-Space Encoding. *IEEE Transactions on Dependable and Secure Computing*, *18*(2), 576–592. https://doi.org/10.1109/TDSC.2019.2903049

Baviera, E., Schettino, G. M., Tuniz, E., & Vatta, F. (2020). Software Implementation of Error Detection and Correction against Single-Event Upsets. *2020 28th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2020*. https://doi.org/10.23919/SOFTCOM50211.2020.9238173

Chandra, T. D., & Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM (JACM)*, *43*(2), 225–267. https://doi.org/10.1145/226643.226647

Chioran, D., & Valean, H. (2020). Arduino based Smart Home Automation System A Simple and Efficient Serial Communication Method. *IJACSA) International Journal of Advanced Computer Science and Applications*, *11*(4). www.ijacsa.thesai.org

Gupta, A., & Charan, C. (2024). Analysis of Universal Asynchronous Receiver-Transmitter(UART). *Proceedings - 2nd IEEE International Conference on Device Intelligence, Computing and Communication Technologies, DICCT 2024*, 194–198. https://doi.org/10.1109/DICCT61038.2024.10532820

Jamdar, V. T., Deosarkar, S. B., & Khobragade, S. V. (2018). An Effective Arduino Based Communication Module for Railway Transportation System. *Proceedings of the 2nd International Conference on Intelligent Computing and Control Systems, ICICCS 2018*, 749–752. https://doi.org/10.1109/ICCONS.2018.8662959

Jeffries, B., Lee, J. A., & Koprinska, I. (2022). 115 Ways Not to Say Hello, World!: Syntax Errors Observed in a Large-Scale Online CS0 Python Course. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, *1*, 337–343. https://doi.org/10.1145/3502718.3524809

Kondaveeti, H. K., Kumaravelu, N. K., Vanambathina, S. D., Mathe, S. E., & Vappangi, S. (2021). A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations. *Computer Science Review*, *40*, 100364. https://doi.org/10.1016/J.COSREV.2021.100364

Kusnadi, A. (2021). Komunikasi Serial dan Metode Pengkodean pada Arduino . *Jurnal Teknologi Dan Sistem Komputer, 9(4), 101-110.*

Lin, L. (M. S. in E. (2020). *Applying Universal Verification Methodology on a Universal Asynchronous Receiver/ Transmitter design*. https://doi.org/10.26153/TSW/11664

Maemunah, M., & Riasetiawan, M. (2018). The Architecture of Device Communication in Internet of Things Using Inter-Integrated Circuit and Serial Peripheral Interface Method. *Proceedings - 2018 4th International Conference on Science and Technology, ICST 2018*. https://doi.org/10.1109/ICSTC.2018.8528663

Martínez-Santos, J. C., Acevedo-Patino, O., & Contreras-Ortiz, S. H. (2017). Influence of Arduino on the Development of Advanced Microcontrollers Courses. *Revista Iberoamericana de Tecnologias Del Aprendizaje*, *12*(4), 208–217. https://doi.org/10.1109/RITA.2017.2776444

Misra, S., Roy, C., Sauter, T., Mukherjee, A., & Maiti, J. (2022). Industrial Internet of Things for Safety Management Applications: A Survey. *IEEE Access*, *10*, 83415–83439. https://doi.org/10.1109/ACCESS.2022.3194166

Muhajir, F., Efendi, S. & Sutarman. (2016). Deteksi Dan Koreksi Multi Bit Error Dengan Partition Hamming Code. *Jurnal Teknovasi, III(2), Pp. 2-3.*

Mukti, R., & Fadilah, A. (2018). Analisis dan Implementasi Hamming Code untuk Sistem Komunikasi . *Jurnal Sistem Dan Informatika, 14(2), 56-63.*

Mythry, S. V., Harsha Vardhini, P. A., Bandaru, T., Gunamgari, S. R., Bandari, K., & Balagoni, N. (2024). The Low Power Implementation of Hamming-Code Encoder and Decoder using GDI Logic for Error Free Transmission and Reception in Digital Data Communication. *2024 2nd International Conference on Computer, Communication and Control, IC4 2024*. https://doi.org/10.1109/IC457434.2024.10486366

Novianti, R. (2019). Prinsip-Prinsip Pengkodean dan Pengoreksian Kesalahan . *Jakarta: Penerbit Universitas Indonesia*.

Nsaif, Y. M., Hossain Lipu, M. S., Ayob, A., Yusof, Y., & Hussain, A. (2021). Fault Detection and Protection Schemes for Distributed Generation Integrated to Distribution Network: Challenges and Suggestions. *IEEE Access*, *9*, 142693–142717. https://doi.org/10.1109/ACCESS.2021.3121087

Potestad-Ordóñez, F. E., Tena-Sánchez, E., Chaves, R., Valencia-Barrero, M., Acosta-Jiménez, A. J., & Jiménez-Fernández, C. J. (2020). Hamming-code based fault detection design methodology for block ciphers. *Proceedings - IEEE International Symposium on Circuits and Systems*, *2020-October*. https://doi.org/10.1109/ISCAS45731.2020.9180451

Qurrata, H. (2018). Implementasi Hamming Code pada Proyek Arduino . *Yogyakarta: Penerbit Andi*.

Rafi, I. (2019). Sistem Komunikasi Data dengan Mikrokontroler Arduino . *Jurnal Teknologi Dan Informatika, 12(2), 89-96*.

Salamea, H. M. T., Torres, D. D. T., Cardenas, P. D. U., & Onate, C. U. (2020). Implementation of the hamming code for the detection and correction of errors in a telerobotic system using an industrial communication protocol. *2020 IEEE ANDESCON, ANDESCON 2020*. https://doi.org/10.1109/ANDESCON50619.2020.9272049

Serror, M., Hack, S., Henze, M., Schuba, M., & Wehrle, K. (2021). Challenges and Opportunities in Securing the Industrial Internet of Things. *IEEE Transactions on Industrial Informatics*, *17*(5), 2985–2996. https://doi.org/10.1109/TII.2020.3023507

Xie, H., Qi, Y., & Alyousuf, F. Q. A. (2023). Designing an ultra-efficient Hamming code generator circuit for a secure nano-telecommunication network. *Microprocessors and Microsystems*, *103*, 104961. https://doi.org/10.1016/J.MICPRO.2023.104961

Yu, J., & Zhang, Y. (2022). Challenges and opportunities of deep learning-based process fault detection and diagnosis: a review. *Neural Computing and Applications 2022 35:1*, *35*(1), 211–252. https://doi.org/10.1007/S00521-022-08017-3